

**A Workflow for Simulation and Visualization
Of Seismic Wave Propagation Using SeisSol,
VisIt and Avizo.**

Thesis by
Luca Passone

In Partial Fulfillment of the Requirements
For the Degree of
Master of Science

King Abdullah University of Science and Technology, Thuwal,
Kingdom of Saudi Arabia

August 2011

The thesis of Luca Passone is approved

Aron Ahmadi
Committee Member

Signature

Date

Markus Hadwiger
Committee Member

Signature

Date

Martin Mai
Committee Chair (Thesis Supervisor)

Signature

Date

**© 2011
Luca Passone
All Rights Reserved**

ABSTRACT

A Workflow for Simulation and Visualization Of Seismic Wave Propagation Using SeisSol, VisIt and Avizo.

Luca Passone

Ground motion estimation and subsurface exploration are main research areas in computational seismology, they are fundamental for implementing earthquake engineering requirements and for modern subsurface reservoir assessment. In this study we propose a workflow for discretizing, simulating and visualizing near source ground motion due to earthquake rupture. For data generation we use an elastic wave equation solver called SeisSol based on the Discontinuous Galerkin formulation with Arbitrary high-order DERivatives (ADER-DG). SeisSol is capable of highly accurate treatment of any earthquake source characterization, occurring on geometrically complex fault systems embedded in geologically complicated earth structures. We then visualize the results with two tools: VisIt (“a free interactive parallel visualization and graphical analysis tool for viewing scientific data”) and Avizo (“The 3D Analysis Software for Scientific and Industrial data”). We investigate each approach, include our experiences from model generation to visualization in highly immersive environments and conclude with a set of general recommendations for earthquake visualization.

ACKNOWLEDGMENTS

I would like to express my gratitude first and foremost to Martin Mai, Associate Professor of Geophysics. If not for him, I would have never had a chance to pursue a thesis in such interesting, exciting and challenging project at a crossroad between geophysics, high performance computing and visualization. His patience and enthusiasm have been the foundations of my work. I would like to extend my gratitude to the other members of the committee, Aron Ahmadi for taking me under his wing in the final stages of the writing by dedicating time and effort far beyond the call of duty, and Markus Hadwiger for his help and extremely constructive criticism of the thesis, his attention to detail is impeccable.

All of this would not have been possible without the incredible help and patience of the team in the visualization laboratories, in particular Christopher Knox, Daniel Acevedo and Iain Georgeson. Without them the project would have never left the workstation and lived the exciting life it did, their support was incredible.

This thesis also owes his existence to the steady stream of chocolates, cookies and extreme understanding of my girlfriend Yveline, who stood by me even after endless “I will be leaving the office in the next 15 minutes” broken promises.

Kirk Jordan played an important role by first introducing me to the project, it is thanks to him that I first begun working with Martin Mai.

I would also like to thank my parents, Christine and Augusto, and my family for “coping” without seeing me for six months at a time. I still remember when I was ten and talking about going to University when I grew up, I hope I have made you proud.

Finally, I would like to thank King Abdullah University of Science and Technology for sponsoring my masters and this research, it has been a great journey and I will always be grateful for the unique experience and incredible people I have had the chance to meet along the way.

TABLE OF CONTENTS

Signatures page	2
Copyright page	3
Abstract	4
ACKNOWLEDGMENTS	5
Table of contents	7
LIST OF ILLUSTRATIONS	10
LIST OF TABLES	11
I. Introduction	12
II. Related Work	14
III. Hardware Resources	15
III.I. Computational facilities	15
III.I.I. Shaheen	15
III.I.II. Nesor Cluster	16
III.I.III. Workstation	16
III.I.IV. Laptop	16
III.II. Showcase Visualization Laboratories	17
IV. Simulation and Visualization Pipeline	18

IV.I. Building a geological model	20
IV.I.I. GAMBIT	20
IV.I.II. Meshing	21
IV.I.III. ParMETIS	22
IV.II. Simulation	23
IV.II.I. Pickpoint distribution script	25
IV.III. Visualization	26
IV.III.I. Post-processing scripts	26
IV.III.II. Visualization tools	29
IV.III.III. VisIt	29
IV.III.IV. Avizo	30
IV.III.V. Avizo and VisIt comparison	31
IV.III.VI. Transfer function	31
IV.III.VII. Visualization portability	35
IV.III.VIII. Validation	35
IV.III.IX. Frame generation	36
IV.III.X. Data import	36
IV.III.XI. Visualization paradigms	37
IV.III.XII. Performance	38
IV.III.XIII. Avizo and VisIt Compared	39
IV.III.XIV. 2D and 3D Video generation	39
V. Reproducing the LOH.4 example	45

V.I. Introduction	45
V.II.Setting up and running SeisSol	47
V.II.I. Mesh generation with Gambit	47
V.II.II. Mesh partitioning	50
V.II.III.Source file	51
V.II.IV.Running SeisSol	51
V.III.Output postprocessing	51
V.IV.Visualization and Video Generation	53
V.IV.I.Introduction	53
V.IV.II.Opening files with VisIt	54
V.IV.III.AESOP video generation	54
V.IV.IV.3D video generation	56
VI.Conclusion	58
GLOSSARY	60
BIBLIOGRAPHY	62

LIST OF ILLUSTRATIONS

1. Workflow from the geological model to the visualization laboratories	19
2. Parallel decomposition of the domain in VisIt.	27
3. Avizo's transfer function editor showing the predefined seismic colormap.	32
4. Visit's transfer function editor showing a transfer function.	33
5. Comparison of two widths for the Gauss opacity transfer function for the w component at depth.	34
6. Point source strike slip viewed at depth for the u, v and w components.	35
7. Comparison between Avizo's graph view, Avizo's tree view and Visit's view for volume rendering of time dependent data of the w component.	37
8. A 41.7 megapixel frame from a movie designed for the AESOP display, showing multiple views for u, v and w.	40
8a. A close up of the four volume renderers.	40
8b. A close up of the four pseudocolor plots of the surface.	41
8c. Rotating volume renders for u, v, w and the magnitude of the uvw vector.	41
9. Aliasing artifacts caused by interlaced stereo rendering for near horizontal straight lines.	42
10. Side by side stereoscopic image of a volume render produced in Avizo.	43
11. Cutaway of the mesh used in the LOH.4 example. Color represents tetrahedron size.	44

LIST OF TABLES

1. ParMETIS performance for domain decomposition run on the workstation.	23
2. Pickpoint distribution example.	25
3. Parallel performance of post-processing script running on the workstation with hot caches.	28
4. Space comparison between AmiraMesh and VTK format.	29
5. Number of frames rendered per minute and per second.	38
6. Summary of Avizo and VisIt's features.	39
7. Material properties for the LOH.4 test.	49
8. P- and S-wave material velocities.	49
9. Growth rate function parameters. “***” indicates no coarsening.	50
10. pickpoints.dat snippets used for postprocessing scripts set up.	52
11. Example of organized directory structure for three example runs.	53

I. Introduction

According to [1], the prediction of near-source ground motion relies primarily on three factors: (1) an appropriate earthquake source characterization that captures the spatiotemporal variation of the rupture process; (2) the accurate calculation of wave propagation through a three-dimensional complex Earth structure; (3) the correct treatment of site effects in the shallow near-surface region underneath the observation site. We claim that this is true from a purely computational point of view, but gaining information and insight from the gigabytes of data produced by such models is essentially the ultimate goal of any scientific simulation. This is of paramount importance when we consider the impact of such estimations: from oil exploration to construction standards, end users and their monetary investments revolve around understanding and applying the computed predictions in meaningful and concrete projects.

Today there are many visualization tools capable of delivering extremely engaging experiences over vast amounts of data: scientist and engineers can dig deep in the areas that interest them the most. As it is common in many fields, there are many approaches, each with their own strength and weaknesses, that can be used to achieve similar results but differing in the amount of effort, investment, interactivity and ease of use. Here we present a summary of our experiences in visualizing fault-rupture models based on simulations generated by SeisSol. SeisSol features many key elements that are important for scientific software: high portability (runs on IBM, Intel and AMD architectures), highly scalable (simulations using up to 65,536 cores have been done on BlueGene/P) and extremely flexible (large numbers of parameters can be adjusted).

SeisSol's output is in the form of seismograms, which provide only a point-like description of the ground shaking. Seeing the dynamics of waves interacting with the local geology and topography, helps to understand the space-time distribution of the shaking pattern and its dependency on the geological structure. To this extent, snapshots in time, or better yet movies, contribute to unraveling the generation and characteristics of seismic wavefields.

We use complicated (i.e. more realistic) excitation functions instead of simple point-source/point-forces to demonstrate the large impact of the space-time evolution of the rupture on the propagating wavefield.

This study starts by introducing hardware resources used for computational and visualization, explaining how they are used and their specifications. The next section is dedicated to analyzing the software tools: it describes roles, usage and where appropriate, performance evaluations and comparison amongst different solutions. We then go on to describe an end to end example from mesh generation to visualization concluding with advice and common pitfalls in earthquake visualization and our outlook for the future.

II. Related Work

Even though modern advances in software and hardware have paved the way for new ways of visualizing data, according to [2], in the scientific visualization community, animation tools are still not widely adopted. Many legitimate reasons exist for this lack of adoption; cost, training, not appropriate for all situations, etc, but the standard static 2D images of PGV (Peak Ground Velocity) and PGA (Peak Ground Acceleration) do not allow for time dependent data to be displayed effectively, on top of this they do not contribute to understanding the complex interactions happening beneath the subsurface. Recently there have been more examples of efforts concerned with visualizing single events [2-5], examples of specialized codes [6, 7] and some captivating animations using off the shelf software such as Maya [2, 8].

One major shortcoming in the aforementioned and in literature is the lack of reusability and reproducibility. Almost all publications do not make scripts and parameters available for the reader to reuse; this makes it very hard to reproduce the experiment, and, in many cases, problems and issues already encountered and solved need to be tackled once again leading to a slow down of the scientific process.

There is also a limited number of efforts concerned with bringing topographical and scientific earthquake data to stereoscopic environments; [9] has done extremely interesting and innovative work in these areas using custom software designed specifically for the KeckCAVE [10], allowing scientists to load and explore multidimensional data in an immersive environment.

III. Hardware Resources

Both computational and visualization of large data sets require state of the art resources: from mesh generation, through the elastic wave solver, post-processing and finally the visualization itself, all parts involved need to be able to cope with the vast data amounts going in, and be capable of providing input required for the next component. An overview of the hardware follows.

III.I. Computational facilities

III.I.I. Shaheen

Shaheen, the fulcrum of the scientific modeling process, is a 16 rack IBM BlueGene/P supercomputer; each contains 1024 quad-core PowerPC 450 compute nodes running at 850MHz with 4GB of RAM, for a total of 64TB.

I/O on Blue Gene/P is provided via quad-core PowerPC 450 I/O nodes also clocked at 850MHz equipped with 4GB of RAM. The 16 racks are further subdivided into four rows: row 0 has 16 I/O nodes per rack whereas rows 1-3 have 8 I/O nodes per rack.

Control of Shaheen is moderated via an IBM Power p550 Express “service node”. The service node contains 8 processing cores operating at 4.2GHz and is provided with 64GB RAM.

With the above configuration, KAUST's Blue Gene/P installation is capable of 220 Teraflops of peak performance[11], placing it at 34th fastest supercomputer in the world[12] providing

all the processing power that SeisSol needs to produce accurate simulations in relative short times.

III.I.II.Neser Cluster

Neser is an Intel Xeon cluster composed of 128 IBM System x 3550 compute nodes. Each node is equipped with 2 quad-core Intel "Harpertown" E5420 CPUs and 32GB of shared physical memory running Novell SUSE Linux Enterprise Server (SLES 10 SP3). Its main focus is to support primary computation with general-purpose pre- and post-processing power.

Neser and Shaheen have high speed connections to the same storage cluster, requiring no data transfer between the two. Each node connects to one of eight switches via 1 Gbit/s ethernet connections, which in turn connect via thirty-two 1 Gbit/s ethernet to main storage and four 1Gbit/s ethernet connections to the metadata server. This means that, if we use VisIt for visualization, once the data is produced by the simulation it never needs to leave the cluster (more on this later).

III.I.III.Workstation

The workstation is mainly used for post-processing and visualization. It runs Fedora 11, with 23GB of RAM, 2 quad core Intel Xenon CPU X5550 clocked at 2.67GHz, Nvidia GPU Quadro FX 4800 with 1.5GB of RAM, 1TB of Hard Disk space and a 23in Dell P2310H monitor with a resolution of 1920x1080.

III.I.IV.Laptop

We use an Apple MacBook with 4GB RAM, 2.2GHz Intel Core 2 Duo, Intel GMA X3100 with 144MB of VRAM and 7200RPM Hard Disk to test Avizo and VisIt's performance on less powerful machines.

III.II.Showcase Visualization Laboratories

The Showcase [13] visualization laboratories offer state of the art equipment for many kinds of visualization, from tiled display walls to 3D stereoscopic displays it caters for many users.

In our work we utilize three of the tiled displays: AESOP, OptiPortal and NexCAVE.

AESOP is a 40-tile display wall featuring 46" NEC ultra-narrow bezel panels, each having a resolution of 1360x768 pixels powered by 10 compute servers, enabling large scale, near-seamless 41.7 megapixel HD images without use of projection. OptiPortal is a scalable 12-tile display of 1920x1080 HD screens powered by 3 compute servers with a total resolution of 24.8 megapixel. The NexCAVE is a 21-tile modular 3D environment system using JVC X-pol LCD displays with 11 computer servers.

All the compute servers have two quad-core i7-950 clocked at 3.07GHz, 6GB of main memory, nVidia GT200b and 10Gb ethernet connections to storage.

Each space will serve different needs as we will see later in the study.

IV. Simulation and Visualization Pipeline

This chapter is concerned with describing the pipeline's components, from the large software components to the scripts that are used to bring them together. For the simulation, we used code for 3D wave propagation that handles complicated, physically realistic earthquake rupture models embedded in 3D geology to allow us to simulate real events. For visualization, we looked at software packages that are multi platform, 3D enabled and capable of running atop our infrastructure.

Figure 1 is used as a guidance to understand this section.

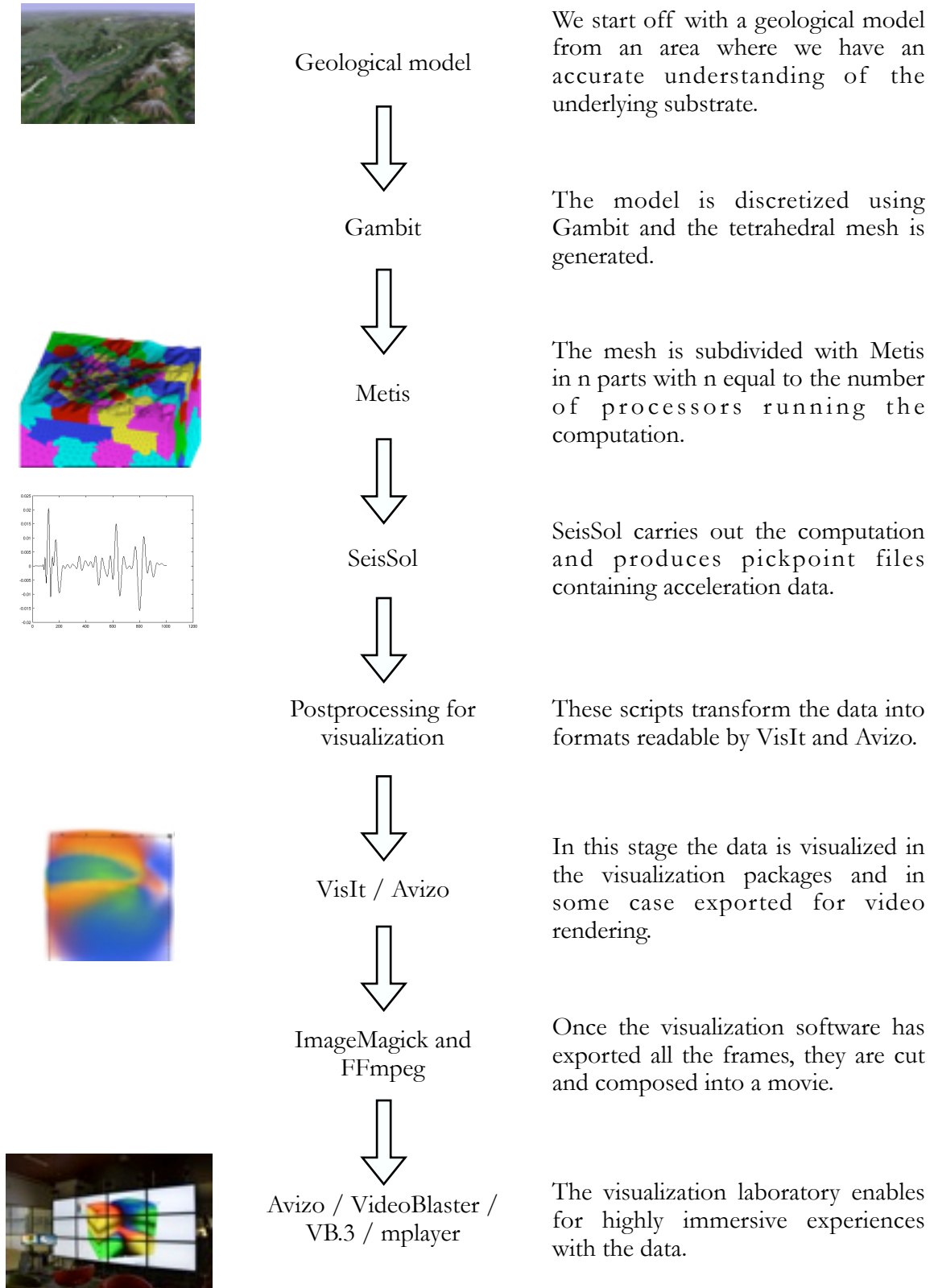


Figure 1: Workflow from the geological model to the visualization laboratories.

IV.I.Building a geological model

The geological model is the first important step for accurate simulation and visualization of an earthquake event: its discretization and subsequent meshing procedure are the foundations of the entire simulation and visualization workflow. The model generally falls into two categories: (1) it is built based on an area where we have an accurate understanding of the substrate and, for example, we want to estimate surface ground motion, (2) it is built on an area for which we have estimates of the geological confirmation and we want to validate their accuracy, used primarily for exploration and reservoir characterization.

Once we have built the model, the subsequent step is meshing. The coarseness of the mesh controls two important aspects of the simulation: accuracy and compute time required. We will see in the section how to find the balance between the two.

Finally we will conclude by explaining how to subdivide the mesh for SeisSol and a performance analysis of an example.

IV.I.I.GAMBIT

GAMBIT is a geometry and mesh generation software for computational fluid dynamics (CFD) analysis [14]. In our workflow it is responsible for geometric construction of the tetrahedral mesh of the domain. It is also capable of mesh quality examination and the very important boundary zone assignment that SeisSol uses for characterizing different materials: the earth subsurface is composed of multiple geological formations, each with different characteristics that are used for parameterizing the computation.

Since the acquisition of Fluent (the company behind GAMBIT) by ANSYS, support has gradually been dropped and plans of moving onto ANSYS ICEM CFD[15] and/or Cubit[16] are undergoing.

IV.I.II.Meshing

The size of the mesh elements is of paramount importance because it affects the two major attributes of a simulation: quality of the output and the time it takes to achieve it. To generate an adequate mesh for the study we need to first calculate the wavelength (λ) of our signal using the wave velocity (c) and frequency (f). This can be simply calculated with the formula:

$$\lambda = \frac{c}{f} \quad [\text{Eq. 1}]$$

We choose the frequency based on the range of interest of our simulation, for our example we are interested only in frequencies below 2Hz. The velocity information of the elastic medium, comes from the geological data; we need to choose between P- and S- wave velocity. In a seismic event, there two main types of body waves[17]: (1) P-waves which have low amplitude and high velocity and (2) S-waves which have high amplitude and low velocity. Since S-waves are slower than P-waves [18], they govern [Eq. 1] as the smaller wavelength requires smaller tetrahedra. Assuming S-wave velocity of 3400m/s the wavelength in the domain of interest is 1700m. According to [19], for a realistic simulation with order 5 accuracy we need 2 elements per wavelength. With this last piece of information we arrive to the conclusion that tetrahedra composing the mesh should have edges no longer than 850m. While generating the mesh, we need to take into consideration a common problem for high order methods: the absorbing boundary condition. By definition, “at absorbing boundaries, no waves are supposed to enter the computational domain and the waves traveling outward should pass the boundary without reflections”[20]. It is well documented in literature that it is very difficult to fulfill this definition in high order methods, especially in corners and for grazing incidence of waves [20-22], where a grazing wave is one which travels almost parallel

to a surface. Attempts at implementing the perfectly matched layered technique from [23] have shown good results but are still not perfect. For this reason, we use a more “brute force” method that leverages on the ability of ADER-DG (a description can be found in section III.III) to use coarsening meshes without suffering from numerical instability. As shown by [24], SeisSol does not suffer from reflections even with very aggressive coarsening parameters. We use this to our advantage and divide our meshes in two areas: the area of interest where the tetrahedra respect the dimensions specified above and where we place the pickpoints (described in section III.IV), and a padding area where the tetrahedra have a very aggressive coarsening function to reduce the amount of overhead computation. The extent of the latter is governed by the speed of the fastest wave and the desired length of the seismogram: if, for example, the area of interest is $2 \times 2 \times 2 \text{ km}^3$, the earthquake is located in the middle, the P-wave travels at 1 km/s and we require a 2 second seismogram, we define a $3 \times 3 \times 3 \text{ km}^3$ domain where the excess part is meshed coarsely.

IV.I.III.ParMETIS

For SeisSol to work in a parallel environment, the mesh needs to be partitioned beforehand. In our workflow ParMETIS, an MPI parallel version of METIS uses multilevel recursive-bisection, multilevel k-way (k is the number of sets), and multi-constraint partitioning schemes [25] for mesh subdivision. Amongst the advantages of using ParMETIS we note that it is extremely fast, provides high quality partitions and is freely available [26]. Table 1 shows an example of a medium to small size layered geology replicating the one used in the LOH.4 test from [1]. It occupies 395MB and contains 945,422 vertices, 5,437,717 tetrahedra.

Parts	ParMETIS time	I/O
128	22.520	1.590
256	24.330	1.610
512	25.900	1.660
1024	27.310	1.610
2048	28.790	1.650
4096	32.320	1.740
8192	32.140	1.660

Table 1. ParMETIS performance for domain decomposition run on the workstation.

Table 1 indicates that even when we increase the number of parts by a factor of 64, the time taken to partition the tetrahedra increases by less than a factor of 1.5. In this particular case we stopped at 8192 partitions since according to [27], each core should have at least 1000 elements to keep efficiency sufficiently high. In our case, a single Blue Gene/P rack (4096 cores) would have been sufficient for the simulation.

One caveat for getting the mesh ready for ParMETIS is that a conversion from GAMBIT to a METIS readable file is required; in the above case an average of 24.4 seconds on top of ParMETIS time was taken by a matlab script to do so.

IV.II.Simulation

For the simulation component, we use SeisSol. SeisSol is an application for simulation of seismic wave propagation in complex 3D media[28] based on the Arbitrary high-order DERivatives Discontinuous Galerkin (ADER-DG) finite-element method[27]. This method was originally introduced in [29], highlighting the excellent wave propagation properties of DG; the ADER-DG approach avoids the Runge-Kutta time integration procedure by substituting it for the far more lightweight ADER approach studied by [30].

SeisSol's workflow follows a four stage process[31]: (1) a data initialization phase reads the input files and a preprocessed mesh; (2) computations are executed spanning multiple time steps and mesh cells; (3) and (4) are concerned with result analysis and output.

For the purpose of this study we summarize the numerical approximation procedure, deferring the reader to [27, 32] for further details. The governing elastic wave equation for propagation in visco-elastic media for three dimensions leads to a first order hyperbolic system of PDEs expressed in the compact form as:

$$\frac{\partial Q_p}{\partial t} + A_{pq} \frac{\partial Q_q}{\partial x} + B_{pq} \frac{\partial Q_q}{\partial y} + C_{pq} \frac{\partial Q_q}{\partial z} = S_p \quad [\text{Eq. 2}]$$

where the vector $Q=(\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{xz}, \sigma_{yz}, u, v, w)^T$ is the vector of the p unknown stresses and velocities related to the corresponding particle motion of the (rock) material acting within each computational cell, A_{pq} , B_{pq} and C_{pq} are the space and material dependent square Jacobian matrices, and S_p is the external source term not dependent on the solution Q . The computational domain is discretized in a conforming tetrahedral mesh of elements $T^{(m)}$ uniquely addressed by the index m . The numerical solution of [Eq. 2] is then approximated in each tetrahedron by polynomials given by a linear combination of time dependent polynomial basis functions $\theta_l(\xi, \eta, \zeta)$ of degree N , and with only time-dependent degrees of freedom $\hat{Q}_{pi}^{(m)}(t)$:

$$(Q_h^{(m)})_p(\xi, \eta, \zeta, t) = \hat{Q}_{pi}^{(m)}(t)\theta_l(\xi, \eta, \zeta) \quad [\text{Eq. 3}]$$

where ξ , η and ζ are the coordinates in a reference element T_E , Q_b denotes the numerical approximation, the index p refers to the numbers of unknowns in the vector Q and l indicates the l th basis function[27]. Further discussion of DG can be found in chapter 1.1 of [33] and [34].

IV.II.I.Pickpoint distribution script

There are two reasons we use a script to define pickpoint location: first to avoid manually defining the thousands of pickpoints required for each run, and secondly to facilitate post-processing by defining them in a specific order. The pickpoints distribution script takes as parameters the size of the domain and the number of pickpoints in each dimension. The output is an ordered list of pickpoints locations evenly distributed in space to be included in the parameter file used to set up the Seisol run. The script executed with the parameters in [Ex. 1] represents a domain stretching $20 \times 100 \times 10 \text{ m}^3$ with eight pickpoints.

```
./pickpoint_distribution.ex -10 10 -50 50 -10 0 2 2 2 [Ex. 1]
```

Table 2 shows how pickpoint distribution in the three dimensional domain.

x	y	z
-10.0	-50.0	-10.0
10.0	-50.0	-10.0
-10.0	50.0	-10.0
10.0	50.0	-10.0
-10.0	-50.0	-0.1
10.0	-50.0	-0.1
-10.0	50.0	-0.1
10.0	50.0	-0.1

Table 2: Pickpoint distribution example.

As we can see, when z is equal to 0 the script substitutes it for -0.1; This is to avoid potential issues when pickpoints are placed exactly at the edge of the domain [35].

IV.III.Visualization

Visualization can be seen as a three stage process: (1) postprocessing the output from SeisSol; (2) achieving the desired visuals through the tools provided by the visualization packages; (3) optional video generation. At each stage there are challenges and best practices, in the following section we will describe how to tackle each stage.

IV.III.I.Post-processing scripts

SeisSol produces three main types of data: fine output, snapshot and receiver output. The parameter file enables fine tuning of the output contents by allowing the user to define the variables to include, the intervals at which they should be written and their format. For the purpose of our visualization needs we use the receiver output. A receiver (or pickpoint) can be positioned arbitrarily in space and records the velocity values u , v and w as they vary in time. SeisSol has a maximum number of pickpoints set to 20,000, therefore if we require higher granularity multiple runs with different pickpoint locations are necessary.

For importing data into VisIt and Avizo we need to use two distinctive file formats: VTK and AmiraMesh. VisIt, unlike Avizo, provides a `visit_writer` library and an entire document titled “Getting Data Into Visit” [36] with over 200 pages dedicated to helping the user getting started. One shortcoming of VisIt is that it is not able to natively process VTK files in parallel: as our data grows in size, we want to be able to distribute the load amongst a larger number of processors to decrease computation time. The only way to achieve this is to do a domain division during the post-processing phase and create a `.visit` configuration file that tells VisIt how to stitch the VTKs back together. We therefore make this assumption: SeisSol’s output will be post-processed using the same number of processors as will be used for visualization. This means that the decomposition process can be run in

parallel (the post processing script for VTKs is MPI enabled) with the only restriction that the number of processes used is less than the number of pickpoint layers.

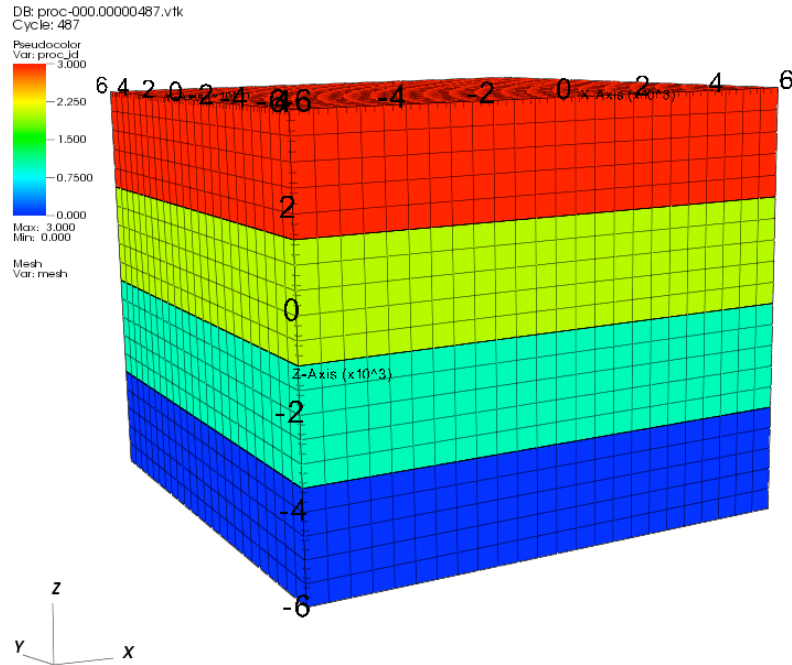


Figure 2. Parallel decomposition of the domain in VisIt.

As we can see in Figure 2, VisIt can display each of the zones assigned to engine processes by using different colors. This is useful as not all domains require the same computation time to complete [37]; it gives the user good visual feedback and allows for fine tweaking of performance by modifying the extents of each zone. Even though the conversion process may seem extremely I/O bound, we can see in Table 3 that parallelizing still has its advantages.

Processors	Open / Close		Keep open		
	Time (s)	E	Time (s)	E	X reference
1	226.63	1.00	37.54	1.00	6.04
2	133.77	0.85	19.84	0.95	6.74
4	75.90	0.75	11.40	0.82	6.66
8	45.00	0.63	8.39	0.56	5.36

Table 3. Parallel performance of post-processing script running on the workstation with hot caches.

“E” represents parallel efficiency for the specific method.

“X reference” is the speedup of “Keep Open” over “Open/Close” using the same numbers of processors. Data shown is for 15,625 pickpoints over 1,000 timesteps.

The post-processing (for both VTK and AmiraMesh) takes a single time step from each pickpoint and writes it to a consolidated file containing all the values for that particular point in time, effectively pivoting the data. We have two options: (1) the main loop opens one pickpoint file at a time, seeks to the last read position, reads the data and closes the file for every time step (the “Open / Close” column in Table 3), (2) if the user is able to change the maximum number of open files, we adjust the operating system limit to a higher value and keep the files open (the “Keep Open” column in Table 3).

The generation of AmiraMesh files does not benefit from parallelization and suffers from the same operating system “limitation”, but has a much smarter way of defining pickpoint locations in space: since in our simulations they are equally spaced, an AmiraMesh file only needs to have the bounding box limits and the number of points along each axis, this saves a considerable amount of space and I/O time. Table 4 shows the space savings that AmiraMesh can achieve over VTK for a visualization output of 15,625 pickpoints with 1500 timesteps.

	Binary(in Megabytes)	ASCII(in Megabytes)	Savings(in %)
AmiraMesh	546	2969	81.6
VTK	816	4300	81.0
Savings (in %)	33.1	31.0	

Table 4. Space comparison between AmiraMesh and VTK format.

IV.III.II. Visualization tools

PGV and PGA maps are excellent tools for estimating a maximum considered event, but to see the dynamics of waves, interacting with the local geology and topography, helps to understand the space-time distribution of the shaking pattern and its dependency on the geological structure. By visualizing earthquakes in high resolution and 3D environments will help to unravel the generation and characteristics of the seismic wavefield. When moving from seismograms to (for example) volume rendering, the observer needs to deal with a whole new set of challenges. These challenges range from transfer function generation to importing data, in this section we focus on how Avizo and VisIt tackle these common problems and the tools available to the observer to analyze the wavefield.

IV.III.II.I. VisIt

VisIt was developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI) to visualize and analyze the results of terascale simulations [38]. Visualization is just part of the VisIt repertoire: data exploration, comparative and quantitative analysis, visual debugging and presentation graphics are all part of the package. VisIt is built on top of 3 main pillars: Qt widget library for the user interface, Python for programmatic scripting and the Visualization ToolKit (VTK) for data storage and visualization algorithms[39]. VisIt is easily extendible with user created plug-ins, it provides a C++ and Java interface to add visualization support to existing applications and is able to read a large amount of mesh types for visualizing both 2D and 3D data. VisIt distinguishes

itself with its distributed and parallel architecture that exploits the Viewer-Engine paradigm: The viewer runs on a local desktop machine and is ideally equipped with a high end graphics card capable of rendering quality visuals; the engine itself can be run either locally or on the same massively parallel machine that produced the data. The latter case has two key advantages: (1) data does not need to be moved to the visualization workstation eliminating the need of vast amounts of local storage and (2) high performance computing resources (thousands of processors and better I/O) can be used for the intensive data processing routines.

IV.III.II.II.Avizo

Avizo, first commercialized in 2007, is a powerful tool for visualizing, manipulating and understanding scientific and industrial data[40]. Avizo is available in 5 editions (Standard, Earth, Wind, Fire and Green) plus an optional eXtension module. The reason for this segmentation is to deliver a tailored user interface and a specific feature set depending on the nature of the data. It also offers a programmatic scripting interface (Tcl), and it is capable of processing very large, out-of core data sets interactively.

Avizo is built on top of Open Inventor [41] which incorporates a very important extension: ScaleViz[42]. ScaleViz allows for tiled display and 3D VR visualization by using a distributed scene graph strategy: a master node is responsible for managing scene graph synchronization and driving the visualization, while the slave nodes run render agents (Open Inventor Render Units) that manage the parallelized GPU accelerated rendering of the image. Although Avizo is not open source nor inexpensive to acquire, its features are very interesting if we need the immensely larger pixel count of a tiled display or want to experience the interactivity and immersive capabilities of a 3D VR environment.

IV.III.II.III.Avizo and VisIt comparison

PGV and PGA maps are excellent tools for estimating a maximum considered event, but to see the dynamics of waves, interacting with the local geology and topography, helps to understand the space-time distribution of the shaking pattern and its dependency on the geological structure. By visualizing earthquakes in high resolution and 3D environments will help to unravel the generation and characteristics of the seismic wavefield. When moving from seismograms to (for example) volume rendering, the observer needs to deal with a whole new set of challenges. These challenges range from transfer function generation to importing data into the visualization tools, and in this section we focus on how *Avizo* and *VisIt* tackle these common problems and the tools available to the observer to analyze the wavefield.

IV.III.II.IV.Transfer function

As highlighted by [43], “The design of the transfer function that maps amplitudes to opacities and colors in the volume rendering algorithms is not an easy task for the interpreter” and small changes have a big impact on the final images. Both *VisIt* and *Avizo* come with a range of presets which offer a good base to start from, but a non negligible amount of time needs to be spent designing a transfer function that is capable of presenting the data effectively. Both offer visual tools that help the user create such functions, but have substantially different qualities. At first first glance the *Avizo* editors looks powerful, but it has major issues when dealing with values of many digits. If we look at Figure 3, we can see that the “Key Value” text box (in the center) can barely fit one digit, and any attempts at modifying the contents truncates an arbitrary amount of digits. This affects the definition of both colors and opacity values. The Location scrollbox somewhat offsets this shortcoming by allowing to pick one of 256 equally spaced points along the color space and alpha gradients, but on the other hand, there is no center point with an even subdivision, requiring

two mirrored points. The interpolation between alpha value points is a very nice feature and allows for smooth gradients without too much refinement (once the key values are set in place) and the histogram provides a useful preview of how much of the data falls under a particular color/gradient combination. Finer control can be achieved by editing the raw colormap file: written in AmiraMesh ASCII format, its content is a four column array with red, green, blue and alpha values and of length equal to the number of points in the “Location” scrollbar.

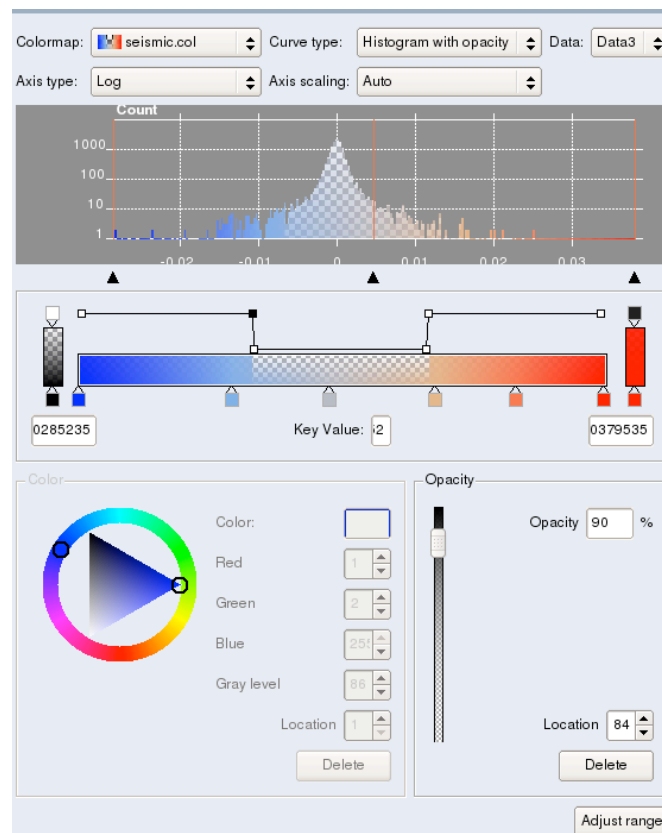


Figure 3. Avizo's transfer function editor showing the predefined seismic colormap.

Figure 4 may lead into believing that VisIt offers much of the same controls as Avizo, but this is hardly true. Starting from the top, we can see how the process of distributing colors is simplified with the help of the “Align” button which distributes the colors uniformly, and an “Equal” checkbox which smoothes the colors by avoiding harsh contrasts. The opacity can

be defined in three ways: (1) loaded from the color table, (2) freeform, (3) Gaussian. The color table tested did not have opacity functions that interested us, the freeform lets the user “draw” by hand on the histogram and although this is not arbitrary it provides a quick way to test out potential candidates. After some visual tests we realized that an inverted Gaussian opacity function fitted our needs very nicely. This can be achieved by using two Gaussian functions with the means at the extremities of the color domain (0 for the left and 1 for the right), and the width (for simplicity VisIt does not use variance) just shy of the middle. Unlike Avizo that provides numeric feedback of each point’s position, VisIt is not as “scientific”, we therefore need to explore the attributes file containing the transfer function information to check for exact symmetry.

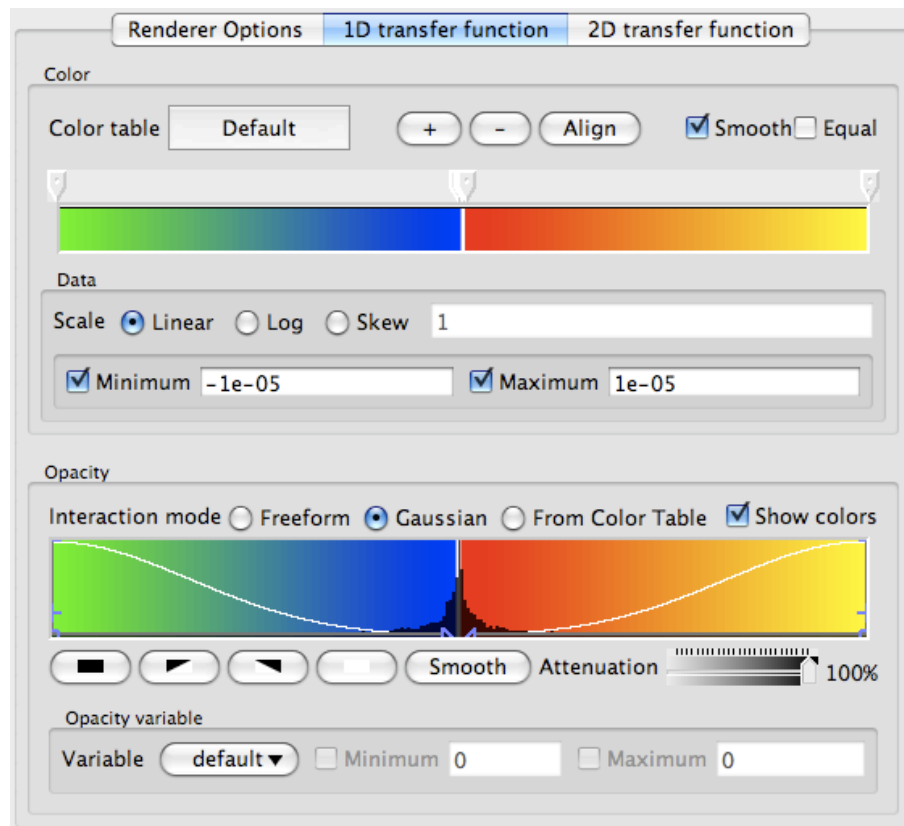


Figure 4. VisIt's transfer function editor showing a transfer function.

The file is in XML format, and rather than containing values at equidistant points it stores information at the user specified points and lets the program do the interpolation. For each Gaussian control point five details are stored: the position of the mean, height, width, x bias and y bias. This allows for very quick modification and enables for exact symmetry around the middle point, in contrast Avizo requires a script to be created if fine control is to be achieved as more points need to be specified.

As mentioned earlier, even small changes to the transfer function have very noticeable effects in the outcome of the visualization. Figure 5 highlights this point by comparing two radically different images, obtained by only changing the width of the Gauss opacity transfer function from 0.495 to 0.5.

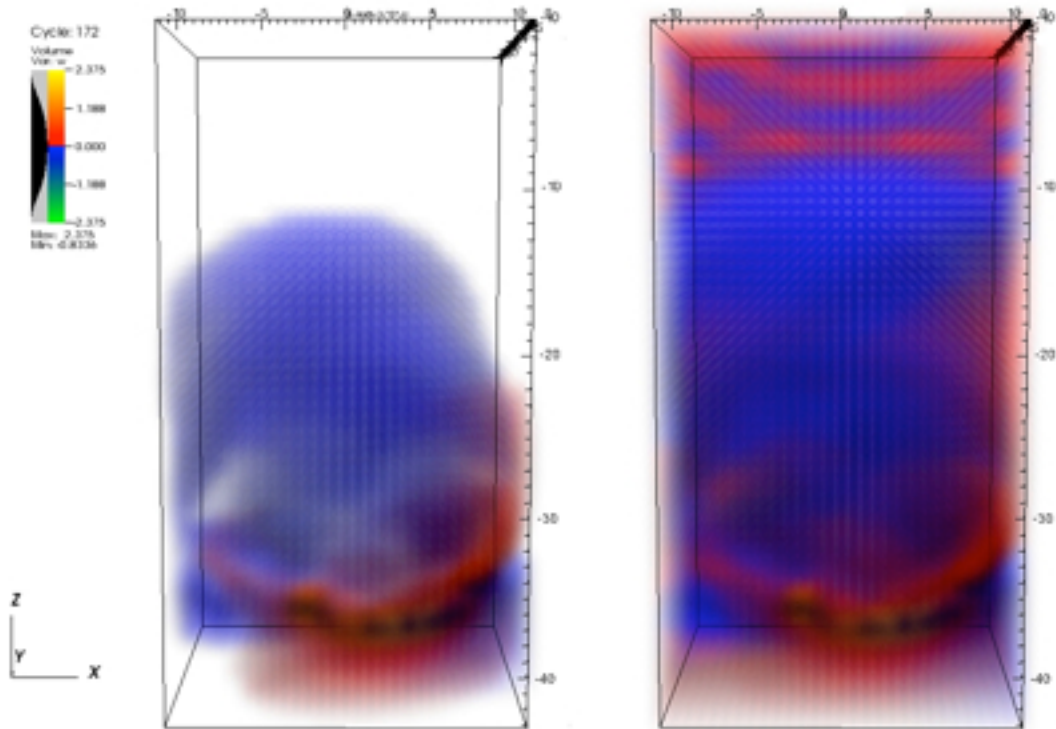


Figure 5. Comparison of two widths for the Gauss opacity transfer function for the w component at depth.

IV.III.II.V. Visualization portability

Since VisIt and Avizo are deployed across different machines, it is important to decouple data from the parameters used to create the visualization. In Avizo, for example, it is possible to export a network template (more on this later), making it very easy to replicate the same visualization achieved on the workstation in the visualization laboratories. This also makes comparing different runs much easier as it is possible to apply the same visualizations and transfer functions defined in precedence. VisIt has a similar function but calls it a session file.

IV.III.II.VI. Validation

Ensuring that we are visually representing the data in the correct manner can be thought of a two stage process: (1) an overall visual inspection can be done to make sure the visualization resembles the intended outcome, (2) pick nodes and analyze the orientation is correct. For this we use a well known source model, a strike slip point source which radiates symmetrically for each component.

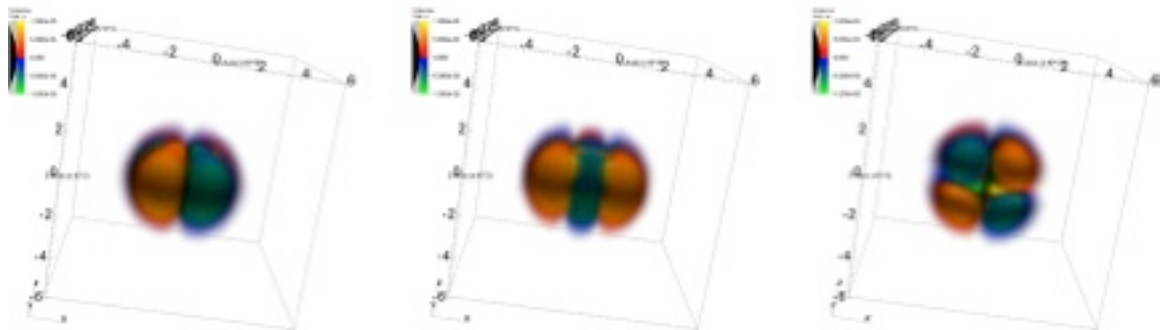


Figure 6. Point source strike slip viewed at depth for the u , v and w components.

Figure 6 shows the radiation amplitude patterns for each component of the 3D vector, and conforms with what we expect from literature [17].

From this we can then use a visual picking tool to extract the value of a node close to the surface. We can then compare the value read from the visualization tool with the raw data from SeisSol to ensure the parameters given to SeisSol and the parameters given to the post-processing tool match in a “end to end” testing manner.

IV.III.II.VII.Frame generation

Basic video frames exporting is easily accomplished by both packages. This can be done in various formats and sizes, but producing a compelling fly over of the data can quickly become a long and tedious process. Avizo and VisIt have orthogonal approaches when it comes to this: the former uses a DemoMaker object which can be used to create Events that act on components, the latter uses a keyframing approach where visit interpolates the frames in between. On top of the above, both offer a scripting interface for maximum flexibility.

Although the Avizo’s model takes a little more time to get used to, it has a much more powerful GUI platform than VisIt; if more fine control is needed then it boils down to a Tcl (for Avizo) versus Python, Java or C++ (for Visit) which is beyond the scope of this study.

IV.III.II.VIII.Data import

Importing data into Avizo was very simple: we picked a format (AmiraMesh) that Avizo supports well, looked at some examples and created a small C program (hoping to parallelize it in the future) that converted from SeisSol output to Avizo input. VisIt was much more difficult to deal with. After some efforts trying to use the prepackaged .silo writer unsuccessfully, we turned to the VTK format. Although VTK files are easy to write and somewhat well documented, the mesh type used within the VTK files affects which plots VisIt is able to visualize. To achieve maximum compatibility, we need to write a curvilinear mesh. This requires the points to be specified in a particular order, and to make post-

processing easier, we need to take this into consideration when creating the PARMETER file for SeisSol and define the pickpoint list accordingly.

IV.III.II.IX. Visualization paradigms

Avizo works by instantiating objects and forming networks between them. Once data is imported, you can right click on the visual object and bring up a menu of all the operations that can be done it. Avizo supports two ways of displaying the networks of objects: (1) tree view and (2) graph view. The graph view is much easier to handle when there is only a limited number of objects, but the edges can quickly become messy and difficult to follow for complicated networks (especially when experimenting). A pane below the network pool shows all the properties for a selected object to enable the user to adjust the parameters.

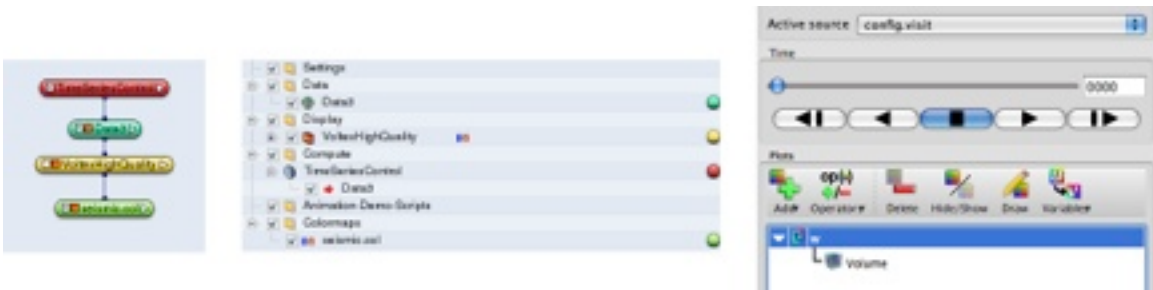


Figure 7. Comparison between Avizo's graph view, Avizo's tree view and Visit's view for volume rendering of time dependent data of the w component.

Visit only supports a tree view of the visualization, but it is simpler and more compact compared to Avizo's. Properties are changed in separate windows brought up by double clicking on the plots or the operators, but even though these settings are hidden away it seems that there are less pixels dedicated to the visualization when compared to Avizo. A great feature that VisIt has over Avizo is the ability to distribute plots over an arbitrary number of windows and "lock" their views and time. For example, if we wish to explore u , v and w together with the same view port at the same moment, we can open three windows and lock them together so a change in any one of them is automatically propagated to the others.

This also becomes extremely useful if we want to test different transfer functions, different plots or any combination of parameters and get instant side by side comparisons.

IV.III.II.X.Performance

Performance evaluation is a very difficult subject. There are many parameters that do not have direct mappings between the two packages, but we try to put together a fair test for both applications and report the result.

The benchmark is based on a 15,625 pickpoint data set simulated in SeisSol over 1500 points in time. The pickpoint files have been converted respectively in AmiraMesh and VTK binary formats for Avizo and VisIt. Almost identical transfer functions have been defined paying special attention to setting matching limits and similar opacity function to recreate near equal workloads. The plot chosen was a volume render with 200 slices viewed directly from above. Compute engine caches were cleared and both programs were set off to try to get through as many time slices as possible on the workstation on a single processor.

	VisIt	Avizo VoltexHighQuality	Avizo Voltex
fpm	1060	44	57
fps	17.67	0.73	0.95

Table 5. Number of frames rendered per minute and per second.

Table 5 suggests that VisIt is considerably faster, which agrees with our feeling from using the softwares daily. A similar test was attempted on the laptop, with VisIt reaching 104 fpm (1.73 fps) and Avizo unable to run due to stability issues. This being said, we cannot help to point out that although the test is informative, it is by no means a definitive benchmark given the very diverse nature of inputs between the two visualizations.

IV.III.II.XI.Avizo and VisIt Compared

In Table 6 we offer a summary of our experiences with the package.

	Avizo	VisIt
Transfer function	Difficult to define, even from the raw color file.	Good GUI and raw control.
Portability	Works on all the high end machines and allows for interactivity on the tiled walls.	Works from low end machines, to highly parallel environments, but not yet capable of running on the tiled walls.
Video generation	Cumbersome but very powerful even from the GUI.	Keyframing is difficult, but the python interface allows for more power.
Data import	Reliable and easy via the use of AmiraMesh, and in some cases saved 33% disk space.	Requires more work to get used to, provides good reference documentation.
Visualization paradigms	Offers network and tree view, depending on user preference.	Offers only tree view.
Performance	Poor	Very good and highly scalable from a low end laptop to Nesar's cluster.

Table 6. Summary of Avizo and VisIt's features.

As we can see, the choice is dictated by the final intended outcome and where the visualization is going to be displayed. On a day to day basis though, most of the time we found ourselves using VisIt.

IV.III.III.2D and 3D Video generation

For the generation of the videos we use a chain of two tools: ImageMagick [44] and FFmpeg [45]. Given the very large resolution of even the smallest of our video walls (twelve HD monitors), it takes many pixels to make effective usage of these facilities. It makes no sense driving such tiled walls with just one stretched HD stream, we could just as well view it sitting down comfortably at a desk, or on a projector. To make the best use of the facilities

we take the following steps: (1) using our visualization software, generate large PNGs per each frame having dimensions proportional to the number of views we want to display and the wall they will be displayed on; (2) With ImageMagick cut the generated PNGs in order to fit them on the appropriate tiles; (3) Use FFmpeg to convert each sequence of PNGs into a separate movie. Figure 8 shows a movie frame designed specifically for the AESOP tile wall with a resolution of 13600 x 3072 pixels (41.7 Megapixel).

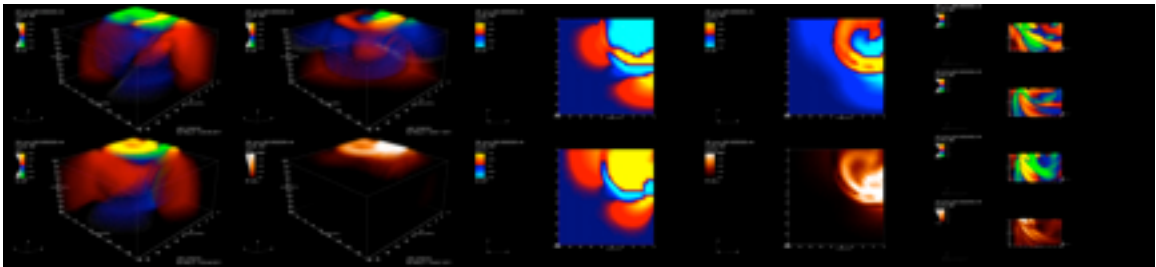


Figure 8. A 41.7 megapixel frame from a movie designed for the AESOP display, showing multiple views for u , v and n .

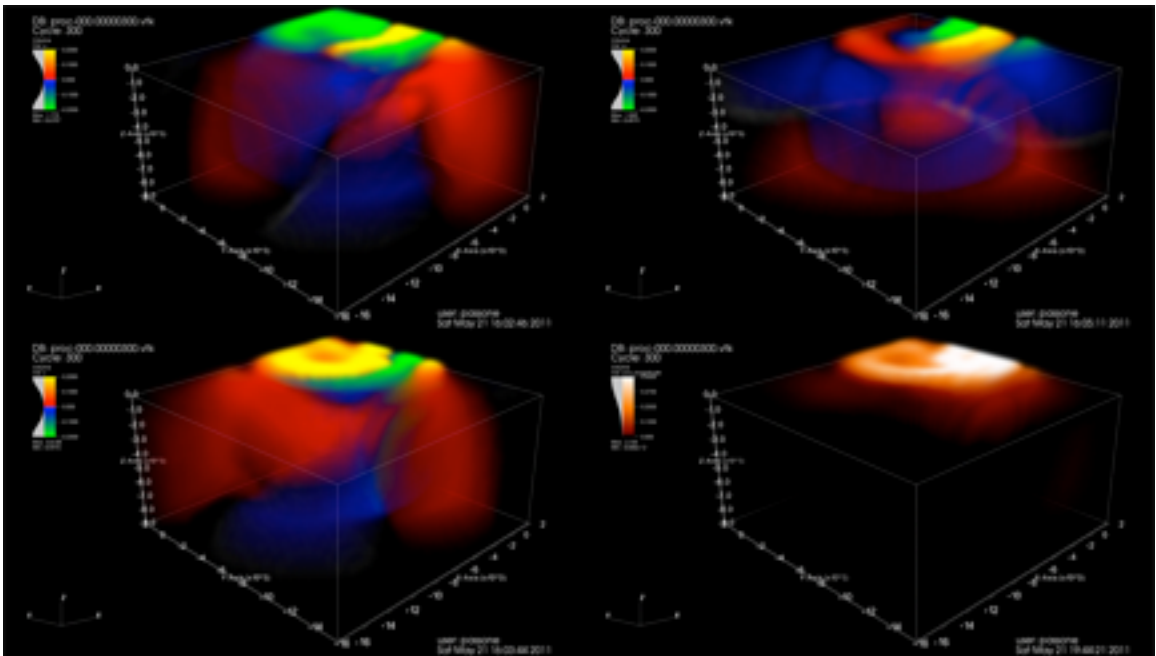


Figure 8a. A close up of the four volume renderers.

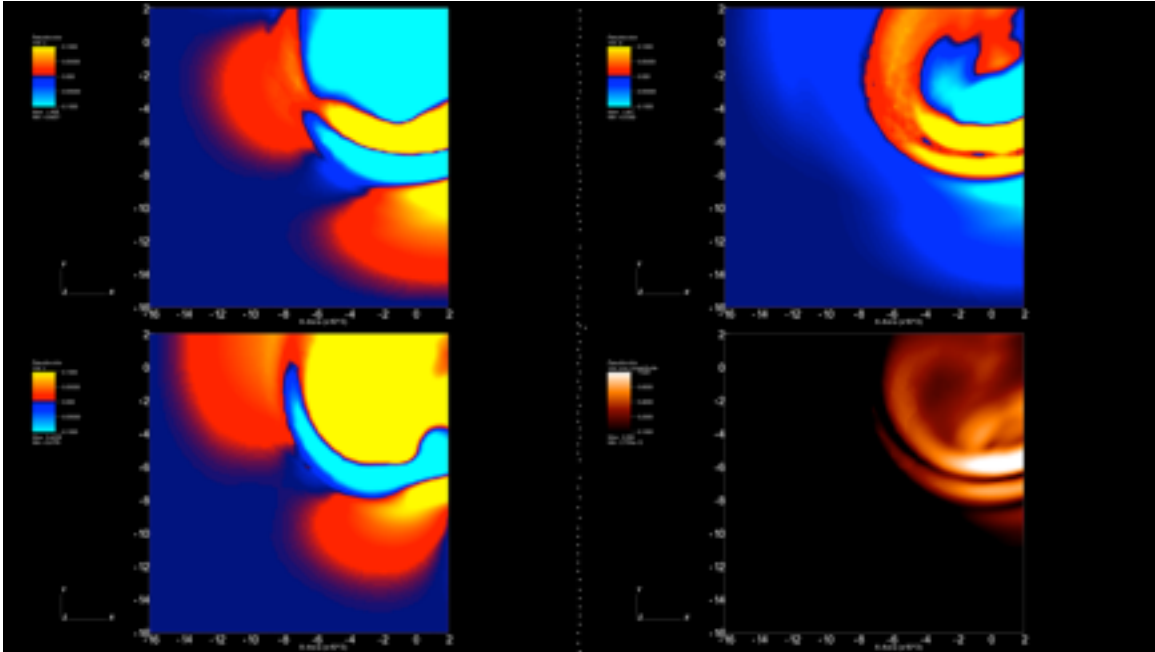


Figure 8b. A close up of the four pseudocolor plots of the surface.

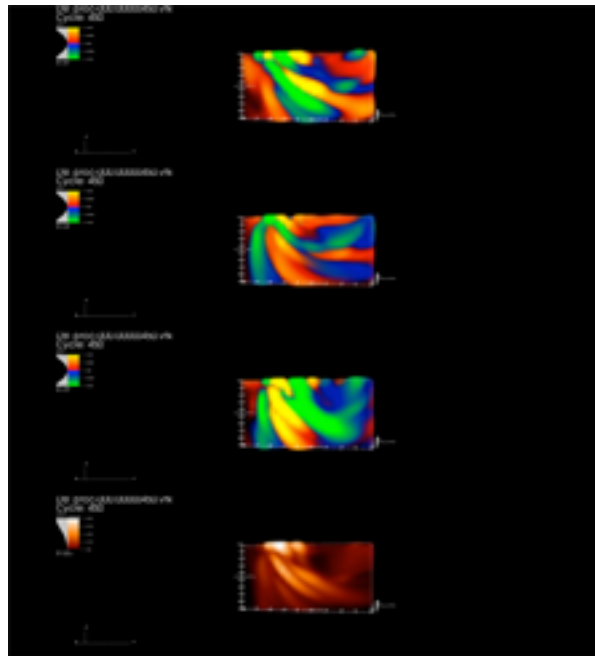


Figure 8c. Rotating volume renders for u , v , w and the magnitude of the uvw vector

On the left side of the frame we have four volume renders (Figure 8a), they represent u , v , w and the magnitude of the uvw vector. The next four (Figure 8b) are pseudocolor surface

plots once again representing u , v , w and the magnitude of the uvw vector. Finally the last column on the right is a rotating view of the first four plots (Figure 8c).

When a 3D movie is required we make use of the NexCAVE. The NEC monitors have alternate pixel rows with opposite polarization sense, with even rows assigned to the left eye and odd rows assigned to the right eye. VisIt can generate separate PNGs for left eye and right eye, which we then interlace using a python script to create a single frame at every time step. It is important that the source frames and the output frame are 1080 rows, as this guarantees perfect alignment with the correct polarized section of the screen. Two issues have become really apparent with this process: (1) it is easier to give the illusions of the visualization being deep into the screen rather than reaching out, (2) when straight lines approach being horizontal they cause very noticeable aliasing (as show in Figure 9).

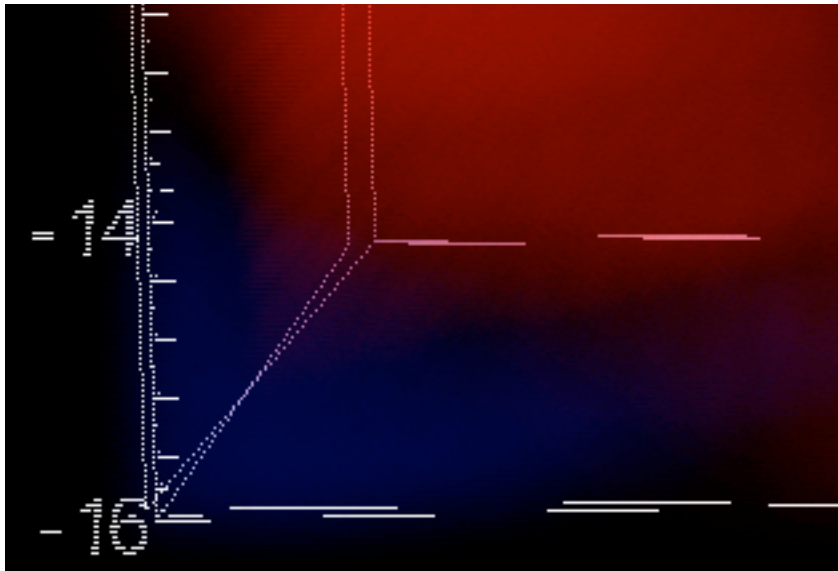


Figure 9. Aliasing artifacts caused by interlaced stereo rendering for near horizontal straight lines.

The former issue is mitigated by refraining from zooming very close into the visualization, and avoiding stretching it beyond the visible area of the screen. For the latter, when setting

up the movie we need to take care that no keyframes cause slow moving viewports with evident aliasing.

Avizo suffers from the same issues, but has a much more difficult set up when it comes to exporting a movie in 3D. Two objects are required to create the frames: MovieMaker and “frame.amov”. The latter should be collaborating with the former, giving the option of generating different flavors of left and right stereo and interlaced output. Unfortunately this does not seem to have any affect on the PNG produced by the MovieMaker object. We decided to modify the python script to alternate between the columns on the left side and columns on the right side of the image in order to achieve similar result to VisIt’s. Figure 10 shows the output of a side by side movie generated in Avizo.

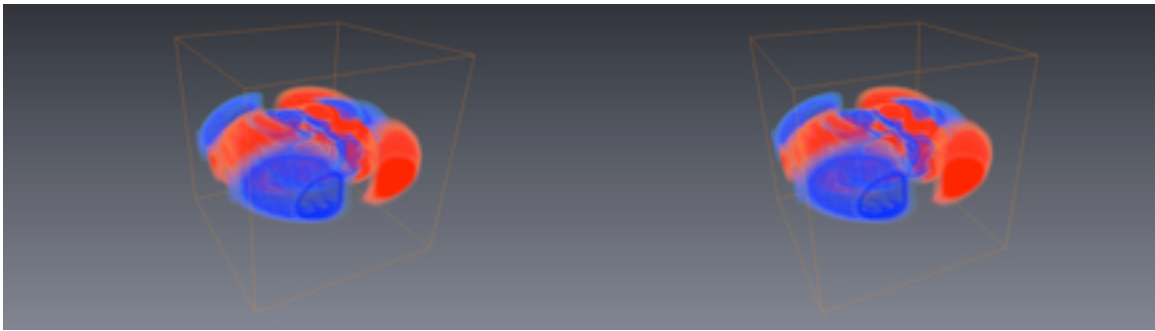


Figure 10. Side by side stereoscopic image of a volume render produced in Avizo

In both cases, the challenge is to make effective use of the computational power available. Since we are working with huge data sets, parallelizing and distributing the computation is very important to complete the process in a useful time frame. For example a 3D movie rendered with VisIt with 15,525 data points for 1000 time frames takes over seven hours to render on a single workstation node. The good thing is that rendering is embarrassingly parallel, and several compute engines can be spawned at the same time with very good strong scaling attributes.

V.Reproducing the LOH.4 example

V.I. Introduction

An increasing number of scientists within the research community have raised warnings regarding the state of practice of reproducible research. A special issue of *Computing in Science & Engineering* [46] was dedicated to raising awareness of tools for addressing reproducibility. In 2010 the Yale Law School formed a Roundtable on Data and Code sharing with the aim to layout recommendations for the community [47]. In 2011, Jarrod Millman organized a mini-symposium at the SIAM Conference on Computational Science & Engineering to further refine standards, tools and techniques [48]. One of the tools mentioned in this space is VisTrails. VisTrails has two key features: it allows for the creation of executable documents [49] and the innate ability to provide provenance [50]. We then began looking at executable documents [51, 52], but experienced difficulties with opening the interactive figures. Only Figure 3 from [53] was capable of delivering the full “executable document” promise. The provenance infrastructure, on the other hand, is extremely powerful and has many desirable features such as variable space exploration, branching of workflows with tagging capabilities and workflow comparison and analysis. All these features are then packed in a very cumbersome interface requiring multiple components just for basic visualization; VisTrails is also missing other important elements such as movie generation and 3D support. We hope these issues will be offset with the ParaView and soon to be released VisIt plugins.

In order to show a fully reproducible experiment, we describe the end to end steps required to visualize an earthquake event based on the LOH.4 example discussed in [1, 54]. Between October 2000 and March 2002, a report was compiled for the Pacific Earthquake Engineering Research Center validating a number of numerical methods for modeling earthquake ground motions over a series of problems. The LOH.4 example is a layer over halfspace test with a propagating thrust dislocation source. The uppermost 1,000 m has $V_s=2,000$ m/s, $V_p=4,000$ m/s, density= $2,600$ kg/m³. The underlying halfspace has $V_s=3,464$ m/s, $V_p=6,000$ m/s, density= $2,700$ kg/m³. For both layers P- and S-wave attenuation (Q) are set to infinite (no attenuation).

The source is a finite fault with strike (ϕ) 115° , dip (δ) 40° and rake (λ) 70° , the fault size is 6 x 6 km and the hypocenter is located at the center of the bottom line of the fault at position (0, 0, -6) km. Using the local fault-plane coordinate system (ζ, η) aligned with strike and dip direction and with its origin at the top northwestern corner of the fault, the hypocenter is located at (ζ_H, η_H) = (3, 6) km. The fault plane is divided into 50 x 50 subfaults of area $A = 1.44 * 10^4$ m² each. The source time function for the slip rate is given by equation (15) and the rupture velocity is constant $v_{rup} = 3000$ m/sec.

In terms of fault-plane basis vectors, the slip vector is:

$$\left[\hat{\xi} \cos(\lambda) - \hat{\eta} \sin(\lambda) \right] S(\xi, \eta, t) \quad [\text{Eq. 4}]$$

where the slip function S has the same shape and amplitude everywhere within the fault surface but is time-shifted by an amount proportional to the distance from the hypocenter. S is given by:

$$S(\xi, \eta, t) = S_0 \left[1 - \left(1 + \frac{\tau}{T} \right) e^{-\frac{\tau}{T}} \right] H(\tau) \quad [\text{Eq. 5}]$$

where H is the Heaviside step function, $T = 0.1$ sec is a smoothing factor controlling the frequency content and the amplitude of the source time function, the static slip S_0 is 1 meter and τ (the time relative to rupture arrival) is:

$$\tau = t - V_{rup}^{-1} \left[(\xi - \xi_H)^2 + (\eta - \eta_H)^2 \right]^{1/2} \quad [\text{Eq. 6}]$$

We will go through the simulation, post processing and visualization set up for the generation of 3D and high resolution movies. Before starting ensure to have a copy of repository [55]; the files mentioned in this section are contained in the LOH4 folder. This example focuses on local visualization and therefore not uses nesar for postprocessing nor visualization.

V.II. Setting up and running SeisSol

V.II.I.Mesh generation with Gambit

First of all we use Gambit to generate the tetrahedral mesh. Figure 11 shows the desired final outcome.

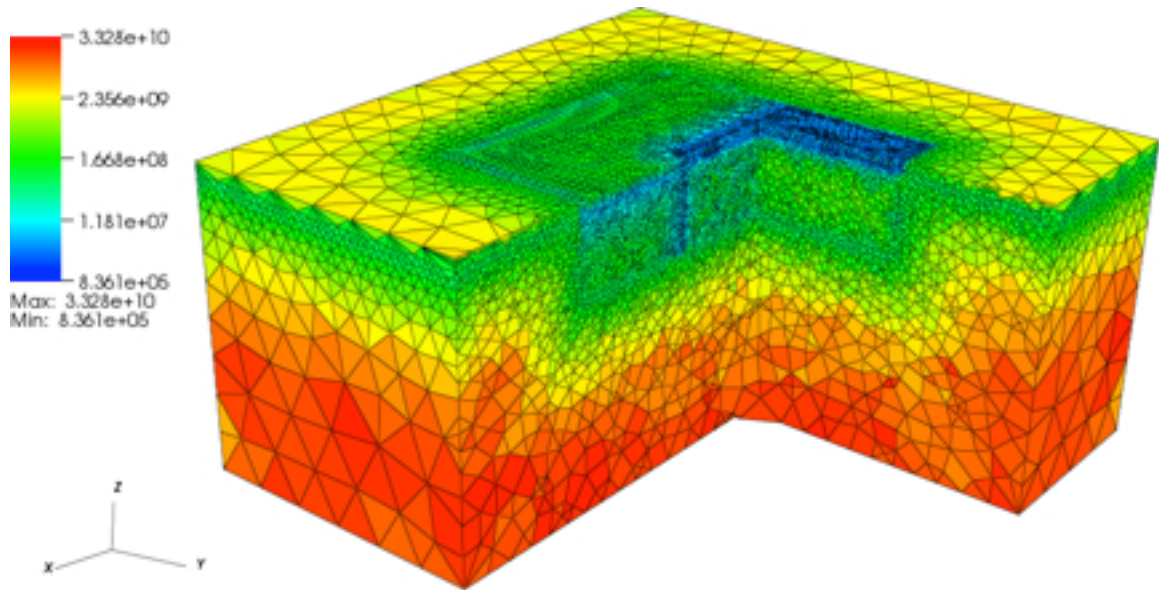


Figure 11. Cutaway of the mesh used in the LOH.4 example. Color represents tetrahedron size.

It is a combination of six volumes: three at the surface 1000 meters thick with fine meshes for higher accuracy, and three deeper volumes with coarsening tetrahedra to reduce computation time. The process can be summarized as follows:

- create the inner top volume (we shall call this small_1);
- create the medium top volume (we shall call this medium_2);
 - proceed to subtract medium_2 from small_1;
- connect the faces that are common between the two volumes.

The final step is important to achieve a conforming mesh. The process is essentially the same for the rest of the volumes, with an extra step required for the large volume in which the faces that interact with both inner volumes need to be split at the boundary.

Once all volumes have been created and the faces have been connected it is time to define the coarsening functions and mesh each volume. As mentioned in the Meshing section, the size of the tetrahedra is governed by the property of the materials. The LOH.4 test has two layers with the properties described in Table 7.

Layer	ρ	μ	λ
1	2600	10400000000	20800000000
2	2700	32400000000	32400000000

Table 7. Material properties for the LOH.4 test.

These properties are used in the the `siv_LOH4.def` file, but we need to extract primary and secondary wave velocities via equations [7] and [8] to determine the size of the tetrahedra.

$$c_p = \sqrt{\frac{\lambda + 2\mu}{\rho}} \quad [\text{Eq. 7}]$$

$$c_s = \sqrt{\frac{\mu}{\rho}} \quad [\text{Eq. 8}]$$

Layer	c_p	c_s
1	4000	2000
2	6000	3464

Table 8. P- and S-wave material velocities.

Table 8 shows the calculated velocities for the LOH.4 example, we now refer back to Eq. 1 to determine the maximum size of the tetrahedra. Since our area of interest is the inner most volumes (`small_1` and `small_2`) we mesh these according to the method discussed in section 3.1.1. and use aggressive coarsening functions for the rest. The final parameters used for meshing are as follows:

Volume	start size	size limit	growth rate
small_1	300	***	***
medium_1	400	1000	1.15
large_1	800	1.4	5000
small_2	350	***	***
medium_2	350	1000	1.15
large_2	800	10000	1.5

*Table 9. Growth rate function parameters. “***” indicates no coarsening.*

There are two more things to take into consideration when defining growth rate functions: first the functions need to be “attached” to surfaces. This lets the growth function know where the smaller tetrahedra start and ensures a conforming mesh. Secondly the numbers above were achieved after mildly altering the parameters in order to avoid highly skewed tetrahedra. Unfortunately for the latter there is no exact solution, but we have found that the parameters above are well suited for this particular example.

Next we need to associate the volumes with the different materials: the three surface volumes with fluid material 1 and the three deeper volumes with fluid material 2.

The last step is to make the three surfaces at the top reflecting (type 101) and the other nine at the edges absorbing (type 105).

Before exporting the mesh ensure that the solver selected is the “Generic” one.

All the commands used in the creation of the mesh are in the LOH4.jou journaling file which stores the entire history of commands.

V.II.II.Mesh partitioning

Next we convert the mesh from .neu to .metis, and split it up into the number of processors that will be used in the simulation. The matlab script neu2metis.m does exactly this. When

launched it asks for the mesh file name and in the conversion process provides the number of tetrahedra contained in the mesh (516,663 in our case). This is important because, as we discussed in 3.2, we need 1,000 tetrahedra per processor to ensure good parallel efficiency; we therefore chose 512 partitions containing each just under 1,010 tetrahedra.

V.II.III.Source file

The last parameter we are going to discuss is the source file which stores the properties of the earthquake event. There are several ways of describing the fault, but in our example we discretize it using 2500 points, we specify their location, strike, dip, rake and onset time followed by a list of acceleration values for each point at each time step (in our case 3,501 time steps at 0.001 intervals). The LOH.4 source is archived at:

```
/project/k77/passone/FSRM_loh4_individual.dat
```

The rest of the parameters are beyond the scope of this study, a working versions can be found in the LOH4 folder in [55].

V.II.IV.Running SeisSol

Assuming we have a working SeisSol installation, the next step is to gather all the files created in the previous steps into a single folder on Shaheen, and run the simulation. When defining the loadleveller script we need to define exactly the same number of processes as the number of partitions created with gambit. An example (batch_LOH4) can be found in the LOH4 folder from the repository.

V.III.Output postprocessing

As mentioned in section 3.4 the output from SeisSol needs postprocessing to be visualized. We assume that the visualization is going to be run locally, therefore the user has copied the pickpoints files to the local machine. The process for both Avizo (AmiraMesh format) and

VisIt (VTK format) starts by creating the pickpoints.dat configuration file; to create it we simply append the pickpoints filenames to the output previously generated by the pickpoints distribution script. For our example, this will create a file with the following format:

line	Value
1	-16000 2000 -16000 2000 -9000 0
2	25 25 25
3	-16000.000000 -16000.000000 -9000.000000
4	-15250.000000 -16000.000000 -9000.000000
5	-14500.000000 -16000.000000 -9000.000000
...	...
15626	1250.000000 2000.000000 -0.1
15627	2000.000000 2000.000000 -0.1
15628	LOH4_0_-pickpoint-00001-00215.dat
15629	LOH4_0_-pickpoint-00002-00215.dat
...	...
31249	LOH4_0_-pickpoint-15622-00321.dat
31250	LOH4_0_-pickpoint-15623-00320.dat
31251	LOH4_0_-pickpoint-15624-00320.dat
31252	LOH4_0_-pickpoint-15625-00320.dat

Table 10. pickpoints.dat snippets used for postprocessing scripts set up.

A makefile in the “post_processing” folder takes care of compiling the scripts that generate VTKs and AmiraMesh files. A parallel version of the VTK generation scripts can be obtained by invoking make with “PAR=1” option; more details can be found in the folder’s README. It is worth mentioning that the scripts will write to folders called “vtk” and “amiramesh” above the source folder specified in the command line, we therefore advise a user to have a directory named “runs” structured as follows:

```

|-LOH4
    |-amiramesh
    |-raw
    |-vtk
|-LOH_surface
    |-amiramesh
    |-raw
    |-vtk
|-point_source
    |-amiramesh
    |-raw
    |-vtk

```

Table 11. Example of organized directory structure for three example runs.

The raw folder contains the raw output from SeisSol, whereas the amiramesh and vtk folders will be automatically generated upon script execution and will contain the file used for visualization. To start the postprocessing it is sufficient to point the script to raw folder (which at this point should contain the pickpoints files from SeisSol and the pickpoints.dat previously generated):

```
./curv_mesh_3d.ex /path/to/raw/ [Ex. 2]
```

V.IV.Visualization and Video Generation

In this section we describe how to visualize and create movies for the AESOP and the JVC 3D polarized displays. For the example we chose VisIt for the visualization, but the frames can be generated with any tool as long as the dimensions are respected.

V.IV.I.Introduction

Before starting the visualization and creating the videos, there are some key points to keep in mind that will save plenty of time later on.

First of all, planning the visualization is very important, especially for the AESOP; it comprises of 40 monitors, therefore spending some time in front of it sketching out the desired layout will help determine the resolution of each video and its contents.

Secondly, the choice of colors is a key factor for the “pleasantness” and usability of the simulation: VisIt, for example, uses a white background by default; although this is acceptable on a single monitor, it dazzles the user when extended over a large area.

When generating a 3D movie, it is advisable to avoid exceeding the boundaries of the screen as this flattens the effect. It is also encouraged to avoid extreme close ups as this makes the visualization very “deep” and can actually hinder the perception of depth. Lastly, we would like to remind the reader of the aliasing problem discussed in section 3.7.

The above is a collection of lessons learned through trial and error, there are no hard rules defined by the community on the parameters values (be it colors, scale, etc.) to be used in a simulation. The problem with setting such rules is that each run has its own set of unique features that would be very difficult to express without carefully tailoring the parameters.

V.IV.II.Opening files with VisIt

First of all we need to open the files. If we generated the VTKs using the parallel version of the postprocessing scripts, we need to open the file called “config.visit”, otherwise we simply open the file series (by making sure smart grouping is used). At this point is up to the user to formulate a compelling visualization. Good starter files for volume rendering and pseudocolor plots can be found in the “LOH4/visit” folder.

V.IV.III.AESOP video generation

Once we are happy with the transfer function, the viewport and (if any) the key framing we can proceed to export the frames. Go to File->Save Movie, pick new simple movie and click

next. Remove the default MPEG output configuration and create a new one with format PNG and the movie size for that particular viewport of the movie. If, for example, we would like the viewport to extend over two screens side by side, then we would set the resolution to 2720×768 . Clicking “continue” takes us to the “Chose length” screen. Since we are generating PNGs, the “Frames per second” field has no relevance (as each timestep in the vtk’s will correspond to one frame), but frame stride must be set to 1. For the LOH4 example we choose 0 and 1000 for first and last frame values. The next screen is “Chose filename”, we would advise outputting the frames for each video in a separate folder with a meaningful name (e.g. `u_volume`, `v_surface`) in order to better keep track of what has already been generated.

Once the vide generation has completed, the next step is to split the frames that span more than one screen into smaller ones. Two scripts in `vis_tools` (“`tile.sh`” and `tiling.py`) work together to split the images to the format appropriate for each monitor. Following the previous example, we assume that we now have a 1000 frames 2720×768 in a folder called `u_volume`. For this case a call to:

```
./tile.sh u_volume/ 2 1 [Ex. 3]
```

will create in the current directory the folder structure `./screens/u_volume/` and inside there will be 2000 files in the following format:

```
frame0000_000_000.png
frame0000_000_001.png
...
frame1000_000_000.png
frame1000_000_001.png
```

Example 4. Filename convention for tiled frames.

This script was originally intended to be able to take a full 40 monitor image and split it up accordingly, but can also be used (like in this case) for smaller frame parts. This means that

each splitting effort will start naming the files as if they were to be positioned at the top left corner. This causes the next step (video encoding) to require some manual effort. The template for encoding is as follows:

```
ffmpeg -r 15 -i ./frame%04d_<column>_<row>.png -qscale 1 avi/
screen_<column>_<row>.avi
```

Example 5. Encoding template for ffmpeg.

Where column and row are three digit 0 padded indexes of the screens. A point to note here is that the first and second set of column and row ids do not need to be the same. This is usually the case when fragmenting smaller images that do not take up the entire wall.

Once the videos have been created they need to be copied across to the vis labs storage facilities in the project folder. At this point we need to first export the configuration script for zone 2 (AESOP display):

```
export CGLX_DEFAULT_CONF=/opt/kaust/config/cglx/z2-csdefault.xml
```

Example 6. Exporting CGLX configuration for zone 2 (AESOP display).

and then launch VB.3 with:

```
/usr/local/cglX/bin/csastart /home/demo/VB3.3 --rows 4 /path/to/folder/with/movies
```

Example 6. Template for starting the visualization in zone 2 (AESOP display).

The movie should now be playing. For troubleshooting please refer to the vb3-tips.txt guide.

V.IV.IV.3D video generation

The 3D video is to be viewed on a single JVC X-pol LCD, therefore we need to create an interlaced image with the left and right eye components interlaced. The process is the same as for the 2D videos until the video generation. Here, in the “Chose format” window we set

the movie size to 1920x1080, tick the “Stereo Movie” option and chose “Left/Right” for the stereo type. Once the frames have been generated, we use the interlacing scripts contained in the repository under the vis_tools folder. The shell script interlace.sh and Python script interlacer.py collaborate in alternating pixel rows from the left image and right image by composing a final output in which the even rows are for the left eye and the odd rows for the right eye (assuming row count begins at 0). The interlace.sh script takes in 4 arguments: start frame, end frame, input directory and output directory, it assumes that the files have the following format: left_frame%04d.png and right_frame%04d.png. The resulting interlaced frames will be written to the output folder in the format: frame%04d.png.

Once the process is complete, we use ffmpeg with the same parameters as before:

```
ffmpeg -r 15 -i ./frame%04d.png -qscale 1 ../video_folder/3D_movie.avi
```

Example 7. template for encoding with ffmpeg

To play the movie follow these steps:

- ensure it is located in /project/ subtree (i.e. not on a local machine).
- ssh into a nexcave machine that is driving the monitors.
- export your preferred display to be used (you can find the screen name by looking at the screensaver) and executing “export DISPLAY=:n.m” with n and m substituted with the values observed in the previous point.
- use mplayer to play the movie by executing “mplayer path/to/movie/3d_movie.avi”

VI. Conclusion

This study demonstrates a pipeline for modeling and visualizing earthquake wave propagation: starting from a well discretized geophysical model, we pass through a number of tools that allow us to visualize a highly accurate prediction of near source ground motion, and ultimately gain information from large amounts of data. With the software and hardware tools we generated and displayed a 41.3 megapixel multi view movie of an earthquake event, which to our knowledge has not been done before, and enabled high pixel count 3D interactive visualization and video generation. In the process of describing the experiment we have highlighted the shortcomings, pitfalls and challenges of moving from an ordinary desktop screen to large video walls and 3D micro-polarized displays.

Our approach enables geophysicists to collaborate, share data, conduct interactive presentations and look for interesting features, patterns and interactions that we have not seen before. The ability to provide many views simultaneously and, in some cases, interactively gives scientist many advantages over the previous visualizations obtained with SeisSol.

There are still some limitations with the current process: for example a single SeisSol run can output maximum 20.000 pickpoints, and processing this many files elegantly is still a challenge, plus having to run multiple simulations to achieve better granularity is not an optimal solution and something we are looking into. Fully automating the pipeline, with automatic transfer function generation has been left for future projects due to time constraints. From the hardware point of view, we are waiting for a new fiber link between

Showcase and the scratch storage subsystem, removing the need to copy any data between HPC and visualization facilities.

Finally, we believe to have paved the way for introducing two new capabilities in the geophysics field: (1) computational steering [56-58] which has been an emerging topic in recent publications, and looks to be poised to deliver interesting and exciting capabilities to scientist; (2) 3D interactive visualization in a “cave” environment. With the former, researchers will be able to intervene and modify the parameters while the simulation is running allowing for a more interactive parameter search exploration, the latter will give scientist the ability to examine results at the same time as they are being produced. The ultimate challenge lies where these two paths cross, and a researcher can steer, split and kill multiple simulations interactively from within a fully immersive stereoscopic environment such as Cornea.

GLOSSARY

AmiraMesh: Amira's native general-purpose file format. It is used to store many different data objects like fields defined on regular or tetrahedral grids, segmentation results, colormaps, or vertex sets such as landmarks. [59]

Elastic wave: motion in a medium in which, when particles are displaced, a force proportional to the displacement acts on the particles to restore them to their original position. [60]

Fault: A fracture in the continuity of a rock formation caused by significant displacement.

Keyframe: a frame containing a drawing that defines the start or end point of a smooth transition.

Pickpoint: it enables for continuous recording of the variables at a particular point in space at specific time intervals.

PGA (Peak Ground Acceleration): maximum absolute value of acceleration found for a particular strong motion record.[61]

PGV: (Peak Ground Velocity): maximum absolute value of velocity found for a particular strong motion record

Stereoscopy: three-dimensional vision produced by the fusion of two slightly different views of a scene.

VTK: The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing and visualization.[62]

A copy of the scripts, resource and configuration files can be found at:

<https://github.com/Pass1/thesis>

BIBLIOGRAPHY

1. Käser, M., P.M. Mai, and M. Dumbser, *Accurate calculation of fault-rupture models using the high-order discontinuous Galerkin method on tetrahedral meshes*. Bulletin of the Seismological Society of America, 2007. **97**: p. 1570-1586.
2. Chourasia, A., S. Cutchin, and B. Aagaard, *Visualizing the ground motions of the 1906 San Francisco earthquake*. Computers & Geosciences, 2008. **34**(12): p. 1798-1805.
3. Furumura, T., B.L.N. Kennett, and K. Koketsu, *Visualization of 3D wave propagation from the 2000 Tottori-ken Seibu, Japan, earthquake: Observation and numerical simulation*. Bulletin of the Seismological Society of America, 2003. **93**(2): p. 870-881.
4. Stupazzini, M., R. Paolucci, and H. Igel, *Near-Fault Earthquake Ground-Motion Simulation in the Grenoble Valley by a High-Performance Spectral Element Code*. Bulletin of the Seismological Society of America, 2009. **99**(1): p. 286-301.
5. Aagaard, B., et al., *Ground-Motion Simulations of Scenario Earthquakes on the Hayward Fault*. 2009. Medium: ED; Size: PDF-file: 15 pages; size: 2 Mbytes.
6. Yu, H., K.-L. Ma, and J. Welling, *A Parallel Visualization Pipeline for Terascale Earthquake Simulations*, in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. 2004, IEEE Computer Society. p. 49.
7. Ma, K.-L., et al., *Visualizing Very Large-Scale Earthquake Simulations*, in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. 2003, ACM. p. 48.
8. Chourasia, A., et al., *Visual insights into high-resolution earthquake simulations*. IEEE Computer Graphics and Applications, 2007. **27**(5): p. 28-34.
9. Elliott, A. *The UC Davis KeckCAVES and earthquake studies: The Trembling Earth*. 2011 [cited 2011 15 August]; Available from: <http://tremblingearth.wordpress.com/2011/06/23/ucd-keckcaves-and-quakes/>.
10. UC Davis. *start [KeckCAVES]*. [cited 2011 15 August]; Available from: <http://keckcaves.org/>.
11. KAUST. *Shaheen*. 2010 Sat May 1 10:39:00 2010 [cited 2011 23/04]; Available from: <http://www.hpc.kaust.edu.sa/documentation/shaheen/>.
12. 500, T. *TOP500 List - November 2010 (1-100) | TOP500 Supercomputing Sites*. 2010 November 2010 [cited 2011 April 2011]; Available from: <http://www.top500.org/list/2010/11/100>.
13. Showcase. *KAUST Visualization Laboratory*. [cited 2011 03 May]; Available from: <http://www.vis.kaust.edu.sa/facilities/showcase/>.

14. IU. *What is GAMBIT, and where is it installed on the TeraGrid? - Knowledge Base*. 2010 23 March 2010 [cited 2011 26 April]; Available from: <http://kb.iu.edu/data/axyy.html>.
15. ANSYS. *ANSYS ICEM CFD Mesh Generation Software*. [cited 2011 26 April]; Available from: <http://www.ansys.com/Products/Other+Products/ANSYS+ICEM+CFD>.
16. csimsoft. *Computational Simulation Software*. [cited 2011 26 April]; Available from: <http://www.csimsoft.com/cubitfeatures.html>.
17. Stein, S. and M. Wysession, *Introduction to Seismology, Earthquakes and Earth Structure*. 2002: Wiley-Blackwell 498.
18. Marshall, P.D., *An Introduction to the Theory of Seismology, 4th Edition - Bullen,KE, BOLT,BA*. Nature, 1986. **319**(6056): p. 814-814.
19. Käser, M., V. Herrmann, and J. de la Puente, *Quantitative accuracy analysis of the discontinuous Galerkin method for seismic wave propagation*. Geophysical Journal International, 2008. **173**(3): p. 990-999.
20. Dumbser, M. and M. Käser, *An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes - II. The three-dimensional isotropic case*. Geophysical Journal International, 2006. **167**(1): p. 319-336.
21. Käser, M. and M. Dumbser, *An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes - I. The two-dimensional isotropic case with external source terms*. Geophysical Journal International, 2006. **166**(2): p. 855-877.
22. Zhou, Y. and Z.J. Wang, *Absorbing boundary conditions for the Euler and Navier-Stokes equations with the spectral difference method*. J. Comput. Phys., 2010. **229**(23): p. 8733-8749.
23. Berenger, J.P., *A Perfectly Matched Layer for the Absorption of Electromagnetic-Waves*. Journal of Computational Physics, 1994. **114**(2): p. 185-200.
24. de la Puente, J., J.P. Ampuero, and M. Kaser, *Dynamic rupture modeling on unstructured meshes using a discontinuous Galerkin method*. Journal of Geophysical Research-Solid Earth, 2009. **114**.
25. Karypis, G. *Family of Graph and Hypergraph Partitioning Software | Karypis Lab*. 2011 [cited 2011 26 April]; Available from: <http://glaros.dtc.umn.edu/gkhome/views/metis>.
26. Karypis, G. and V. Kumar, *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*. SIAM J. Sci. Comput., 1998. **20**(1): p. 359-392.
27. Käser, M., et al., *SeisSol – A Software for Seismic Wave Propagation Simulations*, in *High Performance Computing in Science and Engineering, Garching/Munich 2009*, S. Wagner, et al., Editors. 2010, Springer Berlin Heidelberg. p. 281-292.
28. de la Puente, J., M. Käser, and J.M. Gela, *SeisSol Optimization, Scaling and Synchronization for Local Time Stepping*, in *Science and Supercomputing in Europe*, S. Monfardini, Editor. 2009, CINECA, Italy. p. 300-3022.

29. Dumbser, M., *On the Improvement of Efficiency and Storage Requirements of the Discontinuous Galerkin Method for Aeroacoustics*. PAMM, 2003. **3**(1): p. 426-427.
30. Schwartzkopff, T., C.D. Munz, and E.F. Toro, *ADER: A High-Order Approach for Linear Hyperbolic Systems in 2D*. Journal of Scientific Computing, 2002. **17**(1): p. 231-240.
31. Benedict, S., et al., *Automatic Performance Analysis of Large Scale Simulations*, in *Euro-Par 2009 – Parallel Processing Workshops*, H.-X. Lin, et al., Editors. 2010, Springer Berlin / Heidelberg. p. 199-207.
32. de la Puente, J., et al., *Discontinuous Galerkin methods for wave propagation in poroelastic media*. Geophysics, 2008. **73**(5): p. T77-T97.
33. Hesthaven, J.S. and T. Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Vol. 54. 2008: Springer.
34. Dumbser, M. and C.D. Munz, *Building blocks for arbitrary high order discontinuous Galerkin schemes*. Journal of Scientific Computing, 2006. **27**(1-3): p. 215-230.
35. Mai, M., *Pickpoints location (specifically at the edge)*. L. Passone and A. Ahmadi, Editors. 2010: Thuwal. p. Personal email communication.
36. Whitlock, B. *Getting Data Into VisIt*. 2010 [cited 2011 30 April]; Available from: <https://wci.llnl.gov/codes/visit/2.0.0/GettingDataIntoVisIt2.0.0.pdf>.
37. Visitusers.org. *Load Balancing*. 2008 [cited 2011 29 May]; Available from: http://visitusers.org/index.php?title=Load_Balancing.
38. Webmaster. *About VisIt*. 2008 21 April [cited 2011 25 April]; Available from: <https://wci.llnl.gov/codes/visit/about.html>.
39. Visitusers.org. *500words - Visitusers.org*. 2010 29 January [cited 2011 25 April]; Available from: <http://visitusers.org/index.php?title=500words>.
40. VSG. *Avizo | VSG*. 2011 [cited 2011 25 April]; Available from: <http://www.vsg3d.com/avizo/overview>.
41. VSG, *Open Inventor by VSG*. 2011.
42. VSG. *ScaleViz | VSG*. 2011 [cited 2011 25 April]; Available from: <http://www.vsg3d.com/open-inventor/scale-viz>.
43. André Gerhardt, et al., *Requisites of 3-D Seismic Data Volume Rendering*. GOCAD ENSG Conference 1998.
44. ImageMagick. *ImageMagick: Convert, Edit, And Compose Images*. 1999 [cited 2011 03 May]; Available from: <http://www.imagemagick.org/script/index.php>.
45. FFmpeg. *FFmpeg*. [cited 2011 03 May]; Available from: <http://www.ffmpeg.org/>.
46. American Institute of Physics, *Computing in Science & Engineering*, 2009. **11**(1): p. 77.

47. Yale Law School. *Yale Law School | Data and Code Sharing Roundtable*. 2010 [cited 2011 22 July]; Available from: <http://www.law.yale.edu/intellecualife/codesharing.htm>.
48. Millman, J. *Verifiable, reproducible research and computational science*. 2011 [cited 2011 22 July]; A mini-symposium at the SIAM Conference on Computational Science & Engineering in Reno, NV on March 4, 2011.]. Available from: <http://jarrodmillman.com/events/siam2011.html>.
49. VisTrails. *ExecutablePapers - VisTrailsWiki*. 20 May 2011 [cited 2011 22 July]; Available from: <http://www.vistrails.org/index.php/ExecutablePapers>.
50. Silva, C., et al., *Using VisTrails and Provenance for Teaching Scientific Visualization*. Computer Graphics Forum, 2011. **30**: p. 75-84.
51. VisTrails. *ExecutableLatexTutorial - VisTrailsWiki*. 20 May 2011 [cited 2011 22 July]; Available from: <http://www.vistrails.org/index.php/ExecutableLatexTutorial>.
52. Tohline, J.E. *User:Tohline/IVAJ/Levels2and3 - VisTrailsWiki*. 2010; Available from: <http://www.vistrails.org/index.php/User:Tohline/CPM/Levels2and3>.
53. Troyer, M., et al., *The ALPS project release 2.0: open source software for strongly correlated systems*. Journal of Statistical Mechanics-Theory and Experiment, 2011.
54. Day, S.M., et al., *Tests of 3D Elastodynamic Codes*. 2002, PACIFIC EARTHQUAKE ENGINEERING RESEARCH CENTER.
55. Passone, L., *Thesis repository*. 2011.
56. Vetter, J.S. and D.A. Reed, *Real-Time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids*. International Journal of High Performance Computing Applications, 2000. **14**(4): p. 357-366.
57. Wright, H., et al., *Steering and visualization: Enabling technologies for computational science*. Future Generation Computer Systems-the International Journal of Grid Computing-Theory Methods and Applications, 2010. **26**(3): p. 506-513.
58. Wu, Q.S., et al., *System Design and Algorithmic Development for Computational Steering in Distributed Environments*. Ieee Transactions on Parallel and Distributed Systems, 2010. **21**(4): p. 438-451.
59. Amira.zib.de. *File Format: Amira Mesh Format*. [cited 2011 16 July]; Available from: http://amira.zib.de/mol/usersguide/HxFileFormat_AmiraMesh.html.
60. Encyclopædia Britannica. *Elastic Wave*. 2011 [cited 2011 17 July]; Available from: <http://www.britannica.com/EBchecked/topic/182027/elastic-wave>.
61. O'Brien, L.J., J.R. Murphy, and J.A. Lahoud, *The correlation of peak ground acceleration amplitude with seismic intensity and other physical parameters*. 1977. p. Medium: X; Size: Pages: 96.
62. VTK. *VTK - The Visualization Toolkit*. [cited 2011 16 July]; Available from: www.vtk.org.